UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Programa de Maestría y Doctorado en Música

Faculta de Música

IMPLEMENTACIÓN DE REDES GENERATIVAS ADVERSAS PARA LA GENERACIÓN DE GESTOS SONOROS PARA LA COMPOSICIÓN DE MÚSICA ELECTROACÚSTICA

TESINA
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN MÚSICA (TECNOLOGÍA MUSICAL)

PRESENTA:
ELLIOT YAIR HERNÁNDEZ LÓPEZ

TUTOR
DR. HUGO SOLÍS GARCÍA
(Universidad Autónoma Metropolitana unidad Lerma)

CIUDAD DE MÉXICO, JUNIO DE 2024

PROTESTA UNIVERSITARIA DE INTEGRIDAD Y HONESTIDAD ACADÉMICA Y PROFESIONAL
(Graduación con trabajo escrito)

De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la Institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado:

IMPLEMENTACIÓN DE REDES GENERATIVAS ADVERSAS PARA LA GENERACIÓN DE GESTOS SONOROS PARA LA COMPOSICIÓN DE MÚSICA ELECTROACÚSTICA

que presenté para obtener el grado de Maestría es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi programa de posgrado, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de graduación.

Atentamente

ELLIOT YAIR HERNÁNDEZ LÓPEZ
523014559

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En el presente proyecto de investigación se hablará sobre el método computacional desarrollado en el lenguaje de programación Python, que tiene por objetivo principal implementar algoritmos de *Deep Learnin*g para la generación procedural de gestos y texturas sonoras para la composición asistida de música electroacústica. Las redes GAN [*Generative Adversarial Network*] serán primordiales para el desarrollo de este proyecto por qué permite generar sonidos procedurales basándose en una base de datos de sonidos, fungiendo como una suerte de síntesis generativa.

Este proyecto sigue la línea de investigación del desarrollo de C.A.M.E (Composición Automatizada de Música Electroacústica), el cuál es un método computacional que hace uso de inteligencia artificial, en específico de algoritmos de *Machine Learning* (Aprendizaje de Máquina o Aprendizaje Automático) de tipo aprendizaje no supervisado y en conjunto con técnicas de Recuperación de Información Musical [MIR – *Music Information Retrieval*] sirve para clasificar una base de datos de sonidos basándose en los distintos descriptores de audio usados en ese desarrollo, tales como *RMSE, ZCR, On-Sets, MFCC, Spectral Centroi*d y *Mel Spectogram*. A su vez, se hizo uso del algoritmo de *K-Means* o *K-Medios,* el cual "agrupa" los sonidos en *K* grupos basándose en sus características. Dicho agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o clúster, generando un *ARRAY* donde se encuentran las etiquetas de salida que, en otras palabras, son los sonidos ya clasificados. Posteriormente dichos sonidos se usaron para una composición algorítmica haciendo uso de un sistema caótico, usando los valores como un flujo de datos continuos para controlar los parámetros de ciertas características de los sonidos.

Cabe destacar que, a lo largo de los últimos años ha habido un incremento exponencial de las investigaciones que hacen uso de algoritmos de Aprendizaje Profundo aplicado al sonido y no a la parte visual como lo son Dall-E, Midjourney, Disco Difussion, Stable Difussion, entre otros. Esto es algo bastante emocionante por qué, usar o amoldar esta tecnología al sonido, nos permite crear nuevas herramientas que, en este y muchos otros casos, ayudan al autor a generar gestos y texturas sonoras complejas basados en audios que sean de su agrado

para después aplicarlas a sus composiciones. Esta búsqueda de un "asistente" que genere sonidos, improvise en vivo, clasifique sonidos o que sirva para componer o mezclar obras sonoras, se ha visto reflejada en las investigaciones que a continuación se expondrán y que, a su vez, contextualizan la presente investigación dentro del marco de desarrollo de aplicaciones con inteligencia artificial y que, de igual manera, aportan al conocimiento.

Por ejemplo, Otto Castro Solano en su tesis de doctorado propone el uso de herramientas tecnológicas como asistencia a la composición musical, con el fin de confrontar y complementar la escucha del compositor mediante la escucha de máquina. En su investigación se implementa el aprendizaje supervisado, con el propósito de encontrar similitudes tímbricas entre grabaciones de sonidos urbanos (Castro, 2022). Asimismo, Aaron Escobar Castañeda en su tesis de Maestría aborda la relación humano-máquina en la generación de modelos de escucha basados en tecnologías de aprendizaje automático, específicamente, en la creación de un sistema de escucha artificial aplicado al análisis de la improvisación libre. (Escobar, 2018)

SampleVAE, desarrollado por Max Frenzel, es una herramienta que permite varios tipos de generación de nuevas muestras, así como la clasificación de sonidos y la búsqueda de muestras similares en una biblioteca de muestras existente. Esta herramienta consiste, principalmente, en Autocodificadores Variacionales [VAE – *Variational AutoEncoders*][1] con flujos autorregresivos inversos[2] [IAF – *Inverse Autoregressive Flows*] y una red clasificadora opcional. (Frenzel, 2019)

De igual forma, WaveGAN es un proyecto que hace uso de una red generativa adversarial para la síntesis no supervisada de formas de onda de audio en crudo. La arquitectura de WaveGAN se basa en DCGAN. El generador DCGAN utiliza la operación de convolución transpuesta para muestrear iterativamente mapas de características de baja resolución en una imagen de alta resolución. WaveGAN modifica esta operación de convolución transpuesta para ampliar su campo receptivo, utilizando filtros unidimensionales más largos

[1] Los VAE heredan la arquitectura de los autocodificadores tradicionales y la utilizan para aprender una distribución generadora de datos, lo que permite tomar muestras aleatorias del espacio latente. (Stewart, 2021)

[2] Un flujo autorregresivo inverso se adapta a los espacios latentes de alta dimensión. El flujo propuesto consiste en una cadena de transformaciones invertibles, donde cada transformación se basa en una red neuronal autorregresiva. (Kingma, 2016)

de longitud 25 en lugar de filtros bidimensionales de tamaño 5x5, y aumentando la muestra en un factor de 4 en lugar de 2 en cada capa. El discriminador se modifica de forma similar, utilizando filtros de longitud 25 en una dimensión y aumentando el stride de 2 a 4. Estos cambios hacen que WaveGAN tenga el mismo número de parámetros, operaciones numéricas y dimensionalidad de salida que DCGAN. Posteriormente se añade una capa adicional para permitir más muestras de audio. (Donahue et al., 2018)

Dentro de las aplicaciones comerciales podemos encontrar el proyecto *Artificial Intelligence Virtual Artist* (AIVA), creado por Pierre y Vincent Barreauen en 2016. AIVA es una inteligencia artificial capaz de crear composiciones sonoras emocionales para películas, videojuegos, programas de televisión y anuncios. El algoritmo de AIVA es capaz de analizar y componer obras nuevas basándose en las partituras almacenadas en su amplia base de datos, ya que su arquitectura se basa en algoritmos de aprendizaje profundo y aprendizaje por refuerzo. (AIVA s.f.)

También, se encuentra el proyecto de código abierto Magenta Studio el cuál que explora el rol del Aprendizaje Máquina como una herramienta en el proceso creativo sonoro. Magenta Studio es un plugin para el software Ableton Live para canales MIDI. Su funcionamiento se basa principalmente en autocodificadores variacionales y redes neuronales recurrentes[3] [RNN - *Recurrent Neural Network*] para la generación de notas siguiendo la melodía o el ritmo del clip de audio seleccionado.

Esto es solo un poco de las investigaciones/herramientas actuales que se han desarrollado en los últimos años como ya se mencionó anteriormente, pero son un claro ejemplo de dicha necesidad por desarrollar herramientas con inteligencia artificial que sirvan para asistir el proceso creativo. Es por esto por lo que, en esta búsqueda del autor por explorar las posibilidades de generación procedural de gestos y texturas sonoras, se llegó a la idea de desarrollar un método computacional que haga uso de las redes GAN, las cuales se expondrán en el siguiente capítulo.

[3] Una red neuronal recurrente (RNN) es un tipo de red neuronal artificial que utiliza datos secuenciales o de series temporales. Utilizan datos de entrenamiento para aprender y se distinguen por su "memoria", ya que toman información de entradas anteriores para influir en la entrada y salida actuales. (International Business Machines Corporation [IBM], 2021)

1. REVISIÓN TEÓRICA DE LA INTELIGENCIA ARTIFICIAL

1.1 INTELIGENCIA ARTIFICIAL

La Inteligencia Artificial (IA) ha sido uno de los avances tecnológicos más importantes en los últimos años, puede ser vista como una herramienta para mejorar la eficiencia, productividad y la precisión en distintos ámbitos como clasificar cantidades masivas de datos, hacer predicciones a corto y mediano plazo, generar audios, imágenes, texto y video hasta el manejo autónomo de autos con IA, asistencia médica inteligente, finanzas inteligentes y recientemente el uso de la IA para el desarrollo de ciudades inteligentes. Pero ¿Cómo se puede definir a la inteligencia artificial? A lo largo de este capítulo, con el fin de exponer una noción general, se revisará desde un punto de vista de las ciencias de la computación lo que es la IA. Pero antes de empezar, es necesario entender que a pesar de que es una ciencia muy extensa, se puede dividir en 2 grandes subcampos: el Aprendizaje Máquina - *Machine Learning* y el Aprendizaje Profundo - *Deep Learning*. (Véase Figura 1) Una vez teniendo esto en cuenta, se seguirá con la revisión teórica de lo que es la IA.

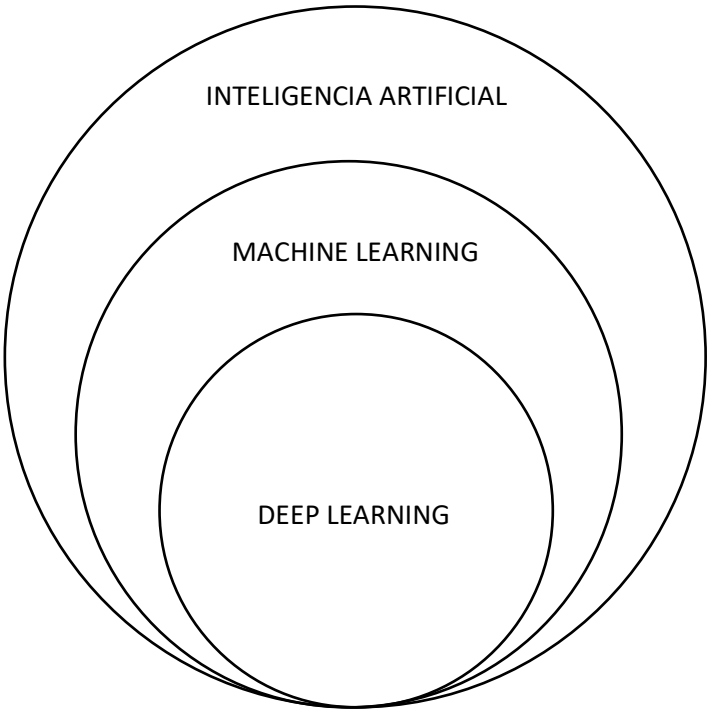


Figura 1. Inteligencia artificial y sus subcampos.

La IA se basa principalmente en 3 grandes escuelas:

1. **Simbolismo**: Se refiere a la IA simbólica que se deriva de la lógica matemática, sugiere que las unidades básicas de cognición de los humanos son símbolos y que la cognición humana es un proceso de razonamiento basado en diversos símbolos. Tanto los humanos como los ordenadores son sistemas de símbolos físicos, los ordenadores pueden utilizarse para simular comportamientos humanos inteligentes. Esta teoría, propuesta por McCarthy en 1956, utilizó por primera vez el término "inteligencia artificial" e hizo contribuciones dominantes al desarrollo del campo, especialmente al éxito de los sistemas expertos.
2. **Conexionismo**: Sostiene que la IA tiene su origen en la biónica, es decir, es el estudio del cerebro humano. Los investigadores creen que la neurona es la unidad básica del pensamiento y no los procesos simbólicos. Es por esto qué, el conexionismo parte de las neuronas y estudia las redes neuronales y los modelos cerebrales.
   Fue hasta la propuesta de redes neuronales simuladas en hardware que resurgió de nuevo la corriente del conexionismo.
3. **Accionismo:** También se conoce como evolucionismo o cibernetismo. Afirma que la IA tiene su origen en la cibernética. Sugiere que la inteligencia depende de la percepción y las acciones, y que la inteligencia no requiere conocimiento, representación o razonamiento. Las primeras investigaciones sobre el accionismo se centraron en la simulación inteligente del comportamiento de las personas. Dio lugar al nacimiento del control inteligente[4] y la robótica en los años 80.

Retomando un poco de la descripción de la escuela del simbolismo, el nacimiento del término *Inteligencia Artificial* se le atribuye a John McCarthy en el año 1956 en la conferencia de Dartmouth. El expone que “La inteligencia artificial es la ciencia y la

[4] La linea de investigación del Control Inteligente es el resultado de la intersección del Control Automático, la Inteligencia Artificial y los sistemas informáticos distribuidos de tiempo real. (Galán et al., 2000)

ingeniería que consiste en crear máquinas inteligentes, especialmente programas informáticos inteligentes" (McCarthy, 2007)

En términos generales, la inteligencia artificial se refiere a los sistemas o las máquinas que imitan la inteligencia humana para realizar tareas y que tienen la capacidad de mejorar iterativamente a partir de la información que recopilan. La inteligencia artificial funciona, combinando grandes cantidades de datos con procesamiento rápido e iterativo y algoritmos inteligentes (redes neuronales), permitiendo al software aprender automáticamente de patrones o características en los datos.

En su forma más simple, la IA es un campo que combina las ciencias de la computación y conjuntos de datos para la resolución de problemas. Abarca los subcampos de Aprendizaje Máquina y el Aprendizaje Profundo (*Deep Learning*), los cuales se componen de algoritmos de IA que buscan crear sistemas que hagan predicciones o clasificaciones basadas en los datos de entrada.

Cabe destacar que hay dos tipos de IA, la IA débil y la IA fuerte. En el caso específico de este proyecto de investigación, se hace uso de la llamada inteligencia artificial débil. La definición y las diferencias de estos dos tipos de IA son:

> La IA débil, también llamada IA estrecha o inteligencia artificial estrecha (Artificial Narrow Intelligence - ANI), está entrenada y enfocada para realizar tareas específicas. La IA débil impulsa la mayor parte de la IA que nos rodea hoy en día. 'Estrecho' podría ser un descriptor más preciso para este tipo de IA, ya que es cualquier cosa menos débil; permite algunas aplicaciones muy robustas, como Siri de Apple, Alexa de Amazon, Watson de IBM y vehículos autónomos.
>
> La IA fuerte se compone de Inteligencia General Artificial (*Artificial General Intelligence* - AGI) y Súper Inteligencia Artificial (*Artificial Super Intelligence*). La inteligencia artificial general (AGI), o IA general, es una forma teórica de IA donde una máquina tendría una inteligencia igual a la de los humanos; sería consciente de sí misma y tendría la capacidad de resolver problemas, aprender y planificar para el futuro. La Super Inteligencia Artificial (ASI), también conocida como superinteligencia, superaría la inteligencia y la capacidad del

cerebro humano. Si bien la IA fuerte sigue siendo completamente teórica sin ejemplos prácticos en uso hoy en día, eso no significa que los investigadores de IA no estén explorando también su desarrollo. Mientras tanto, los mejores ejemplos de ASI podrían ser de ciencia ficción, como HAL, el asistente informático de la película 2001: A Space Odyssey. (International Business Machines Corporation [IBM], 2020)

1.2 APRENDIZAJE MÁQUINA – MACHINE LEARNING

Es un subcampo de la inteligencia artificial que, a través de algoritmos, dota a los ordenadores de la capacidad de identificar patrones en datos masivos para hacer predicciones. Este aprendizaje permite a los computadores realizar tareas específicas de forma autónoma, es decir, sin necesidad de ser programados. Los algoritmos de Aprendizaje Máquina se dividen en tres categorías, siendo las dos primeras las más comunes:

- Aprendizaje supervisado: estos algoritmos cuentan con un aprendizaje previo basado en un sistema de etiquetas asociadas a unos datos que les permiten tomar decisiones o hacer predicciones.
- Aprendizaje no supervisado: estos algoritmos no cuentan con un conocimiento previo. Se enfrentan al caos de datos con el objetivo de encontrar patrones que permitan organizarlos de alguna manera.
- Aprendizaje por refuerzo: su objetivo es que un algoritmo aprenda a partir de la propia experiencia. Esto es, que sea capaz de tomar la mejor decisión ante diferentes situaciones de acuerdo a un proceso de prueba y error en el que se recompensan las decisiones correctas.

  (Iberdrola, 2019).

1.3 APRENDIZAJE PROFUNDO – DEEP LEARNING

Es un subgrupo del Aprendizaje Máquina o Aprendizaje Automático donde un conjunto de métodos y algoritmos permiten a una máquina aprender de manera automática con base en experiencias pasadas.

El funcionamiento principal se basa en redes neuronales artificiales (véase figura 1), donde las redes neuronales artificiales permiten realizar tareas de clasificación en juegos de datos etiquetados y tareas de regresión en juegos de datos continuos, aunque también permiten realizar tareas de segmentación en juegos de datos no etiquetados a partir del establecimiento de similitudes entre los datos de entrada. (Casas et al., 2020)

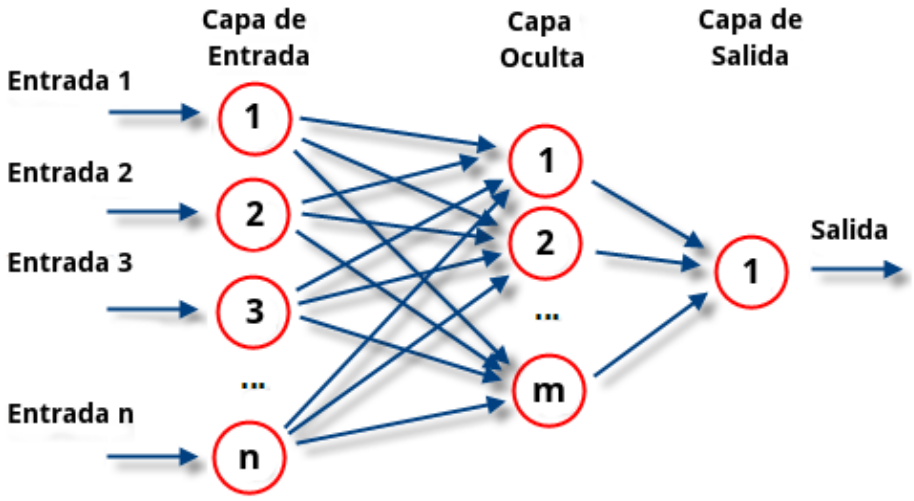


Figura 2. Estructura de una red neuronal. Recuperado de: https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/

En la imagen se pueden observar las capas de entrada que, básicamente, en estas capas es donde va el conjunto de entrenamiento o los datos de entrada, como su nombre lo dice. Estas capas procesan los datos, los analizan o los clasifican y pasan a la siguiente capa.
n representa un número indeterminado de capas.

Las capas ocultas contienen unidades no observables, es decir, el valor de cada unidad oculta es alguna función de los predictores. (International Business Machines Corporation [IBM], 2020)

También, al igual que las capas de entrada, cada capa oculta analiza la salida de la capa anterior, la procesa y la pasa a la siguiente capa.
m representa un número indeterminado de capas.

Al final, se observa la capa de salida, donde ésta capa proporciona el resultado final de todo el procesamiento de datos que realizó la red neuronal. Puede tener una o más salidas, todo depende del problema (si es clasificación binaria o multiclase, por ejemplo).

En otras palabras:

> Las unidades se conectan con fuerzas de conexión variables (o ponderaciones). Los datos de entrada se presentan en la primera capa, y los valores se propagan desde cada neurona hasta cada neurona de la capa siguiente. Al final, se envía un resultado desde la capa de salida.
>
> La red aprende examinando los registros individuales, generando una predicción para cada registro y realizando ajustes a las ponderaciones cuando realiza una predicción incorrecta. Este proceso se repite muchas veces y la red sigue mejorando sus predicciones hasta haber alcanzado uno o varios criterios de parada.
>
> (International Business Machines Corporation [IBM], 2020)

Al principio, las respuestas de una red neuronal son muy poco precisas, ya que las ponderaciones son aleatorias, es por esto por lo que el entrenamiento es tan importante en este tipo de modelos. Lo que se suele hacer, es darle a la red neuronal ejemplos de los que ya se conoce el resultado, se comparan los resultados conocidos con lo que arroja la red neuronal y esta información pasa hacia atrás a través de la red, cambiando las ponderaciones. A medida que el entrenamiento progresa, los resultados de la red son cada vez más precisos en la replicación de los resultados conocidos. Por lo que, cuando la red neuronal ya está completamente entrenada, se puede aplicar para predecir resultados desconocidos.

Este funcionamiento específico de este tipo de algoritmos se aplica para todos los modelos de aprendizaje profundo, tales como las redes neuronales recurrentes [*RNN – Recurrent Neural Network*], redes neuronales convolucionales [*CNN – Convolutional Neural Network*], autocodificadores, perceptrón multicapa y las redes generativas adversas [*GAN –*

Generative Adversarial Network]. Siendo este último, el principal enfoque de desarrollo para esta investigación, el cual se abordará después de exponer teóricamente ¿Qué son las redes neuronales artificiales? Posteriormente, el capítulo 2 se enfocará en la programación, entrenamiento, puntaje y resultados de la red GAN.

1.3.1 REDES NEURONALES ARTIFICIALES

En esta sección, se definirán las Redes Neuronales Artificiales - RNA y se expondrá su funcionamiento y evolución a través de un breve repaso por los momentos más destacados de la historia del desarrollo de las RNA.

Las redes neuronales artificiales (Artificial Neural Networks - ANN), también llamadas sistemas de procesamiento en paralelo (*Parallel distributed processing systems* - PDPs) o sistemas conexionistas (*Connectionist systems*), se pueden definir como "un conjunto de algoritmos inspirados en el mecanismo de comunicación de la neurona biológica" (Casas et. al., 2020). Dichas unidades de procesamiento se organizan en 3 capas: capas de entrada, capas ocultas y capa de salida (Figura 2). Estas estructuras tienen la "propiedad" del aprendizaje, es decir, la adquisición de conocimientos a partir de ejemplos.

> Este aprendizaje se produce mediante un estilo de computación denominado en *paralelo* que intenta simular algunas de las capacidades que posee nuestro cerebro (…) Los tres elementos clave de los sistemas biológicos que pretenden emular son el procesamiento en paralelo, la memoria distribuida y la adaptabilidad. (Torra, 2004)

El funcionamiento general una red neuronal artificial se basa en el modelo básico de una red neuronal biológica: dendritas, axón y cuerpo o soma (Véase figura 3 y 4).

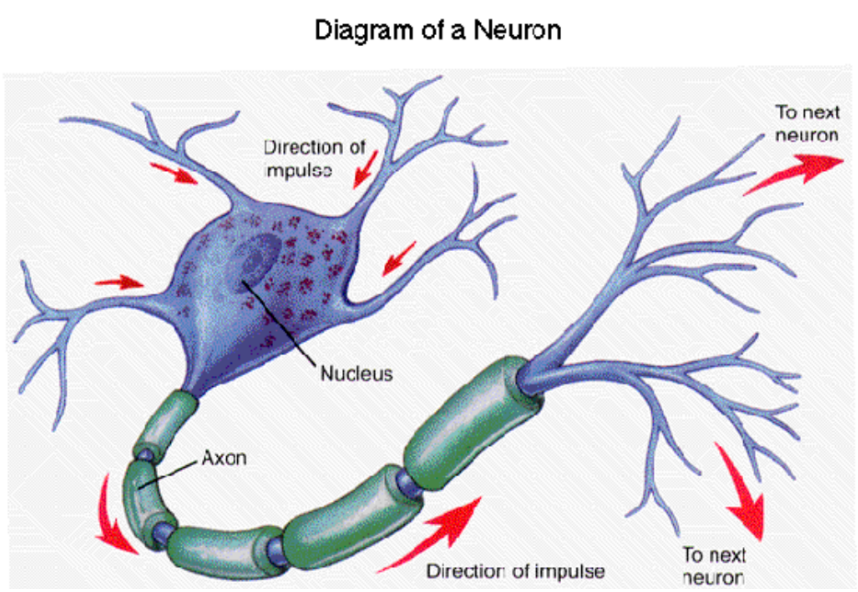


Figura 3. Neurona biológica. Recuperado de:
http://psych.wisc.edu/faculty/pages/croberts/topic3.html

Si hacemos una comparación entre una neurona biológica y una artificial con sus diferentes partes obtendríamos lo siguiente:

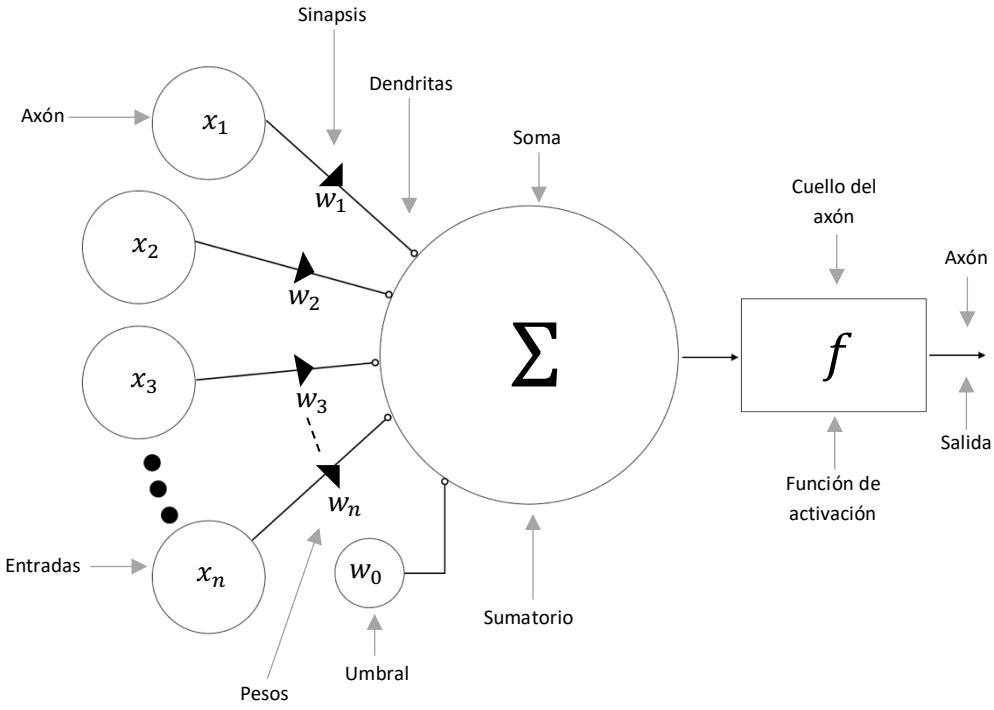


Figura 4. Comparación de neurona biológica y artificial

Dónde la parte superior de la Figura 4 representa las partes ya mencionadas de una neurona biológica y la parte inferior representa las partes de una neurona artificial básica, las cuales son:

- ⇒ Entradas (x_i): Son el estímulo que recibe la neurona artificial.
- ⇒ Pesos sinápticos (w_i): Defina la fuerza de una conexión sináptica entre dos neuronas, la neurona presináptica y la neurona postsináptica.
- ⇒ Umbral (w_0): Es un peso más que es modificado a través el proceso de aprendizaje de la red.
- ⇒ Regla de propagación $(\sum_{i=1}^{n} w_i x_i)$: Determina el potencial de la interacción de la neurona i con las n neuronas vecinas.
- ⇒ Función de activación (f): Determina el estado de activación actual de la neurona en base al potencial resultante y al estado de activación anterior de la neurona.

  Para más información sobre las diversas funciones de activación véase la Figura 22.

⇒ Función de salida (Y): Proporciona el valor de salida de la neurona, en base al estado de la activación de la neurona.
(Areli-Toral, s. f.)

Por lo que dicha estructura de la red neuronal artificial en su forma básica puede representarse de la siguiente forma:

$$Y = f(w_o \sum_{i=1}^{n} w_i x_i)$$

Como se mencionó anteriormente, los modelos matemáticos intentan replicar la forma biológica de transmisión de las señales en las neuronas. Por ejemplo, las dendritas envían señales eléctricas de entrada al cuerpo celular de la neurona. Una señal de entrada más fuerte dispara la señal de salida de la neurona a través del axón y a su vez, esta señal de salida puede actuar como señal de entrada para múltiples neuronas y así sucesivamente.

Originalmente, las redes neuronales artificiales fueron pensadas para la construcción de una máquina que se comportara igual que el cerebro humano, dado que las capacidades de las redes neuronales artificiales (en ese entonces hechas con compuertas lógicas electrónicas) son más rápidas que las biológicas, debido a que una neurona artificial tarda nanosegundos en reaccionar ante un estímulo y una neurona biológica tarda milisegundos en reaccionar. Sin embargo, es acertado destacar que el cerebro al ser sumamente complejo en su funcionamiento es capaz de resolver problemas que las computadoras aún no pueden debido a las miles o millones de células interconectadas. Según Kieran Greer "La interconexión masiva y jerárquica del cerebro parece ser la condición previa fundamental para la aparición de la conciencia y el comportamiento complejo." (Greer, 2012, pp. 47)

Según Xabier Basogain, las principales características de una RNA o ANN son:

⇒ Aprender: adquirir el conocimiento de una cosa por medio del estudio, ejercicio o experiencia. Las ANN pueden cambiar su comportamiento en función del entorno. Se les muestra un conjunto de entradas y ellas mismas se ajustan para producir unas salidas consistentes.

⇒ Generalizar: extender o ampliar una cosa. Las ANN generalizan automáticamente debido a su propia estructura y naturaleza. Estas redes pueden ofrecer, dentro de un margen, respuestas correctas a entradas que presentan pequeñas variaciones debido a los efectos de ruido o distorsión.

⇒ Abstraer: aislar mentalmente o considerar por separado las cualidades de un objeto. Algunas ANN son capaces de abstraer la esencia de un conjunto de entradas que aparentemente no presentan aspectos comunes o relativos
(Basogain, 2008)

Los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch y Walter Pitts. Ellos plantearon una teoría acerca de la forma de trabajar con las neuronas en su artículo *A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY* (1943), incluso, ellos modelaron una red neuronal simple mediante circuitos eléctricos. Sin embargo, dicho modelo neuronal se vio limitado debido a las condiciones técnicas de esos años.

Aunado a esto, en 1949, Donald Hebb fue el primero en explicar los procesos del aprendizaje -que es el elemento básico de la inteligencia humana- desde un punto de vista neuropsicológico, desarrollando una regla de cómo es que ocurre tal aprendizaje, a esta base biológica se le llama plasticidad neuronal, este concepto hace referencia a la capacidad que tiene el sistema nervioso de modificar la naturaleza y la fortaleza de las sinapsis. Según Arturo Torres, la hipótesis que defiende Hebb:

> "es que nuestro cerebro almacena información en redes neuronales. La estructura del sistema nervioso y las relaciones entre sus elementos constituyen la información que procesamos […] si el axón de la neurona A está suficientemente cerca de la célula B y contribuye de forma repetida a dispararla, ciertos cambios estructurales o metabólicos aumentarán la eficacia de tal sinapsis". (Torres, 2017)

A esto mencionado, se le conoce como asamblea celular, de esta forma, cuando esto ocurre de forma repetida y entre más fuerte sea la contingencia entre la activación neuronal a un determinado tipo de estímulo, la probabilidad de que estas mismas redes neuronales o

asambleas celulares se disparen ante estos mismos impulsos incrementa. En otras palabras, el aprendizaje propuesto por Hebb se basa en la consolidación de sinapsis preexistentes ante determinados estímulos. En la actualidad, éste es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal, por lo qué, los trabajos de Hebb formaron las bases de la teoría de las redes neuronales, pero, no solo eso, esta teoría ayudó a que las redes neuronales artificiales se basaran en el proceso de una red neuronal biológica, transmitiendo señales mediante impulsos eléctricos, como se mencionó al principio de este capítulo.

Siguiendo con la historia del desarrollo de las redes neuronales artificiales, otro evento sumamente destacable fue, cuando en 1957, Frank Rosenblatt comenzó el desarrollo del perceptrón, que es la red neuronal más antigua, con uso, hoy en día, en aplicaciones como identificador de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado en el entrenamiento. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función XOR y, en general, era incapaz de clasificar clases no separables linealmente.

Fue en 1969 cuando Marvin Misky y Seymour Papert publicaron su libro llamado *Perceptrons,* en el cuál probaron que los perceptrones eran esencialmente un tipo de modelo lineal capaces de procesar únicamente clasificaciones lineales, por lo que el modelo del perceptrón era muy débil, dado que las funciones no lineales son extensamente empleadas en computación y en los problemas del mundo real. Un par de años después, Paul Werbos (1974) desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*).

Si bien se han logrado avances significativos en la comprensión y aplicación de las redes neuronales artificiales, es un campo en constante evolución y crecimiento. Los investigadores continúan explorando nuevas arquitecturas, algoritmos y aplicaciones para mejorar la capacidad de las RNA en una amplia variedad de tareas de aprendizaje profundo y procesamiento de datos. Desde la creación de modelos como el perceptrón en los años 50 hasta los desarrollos más recientes como las redes neuronales convolucionales (CNN),

las redes neuronales recurrentes (RNN) o las redes generativas adversas (GAN), las cuáles se abordarán en la siguiente sección de este capítulo.

La historia de las RNA es testigo de un progreso continuo y fascinante, pero por cuestiones de extensión, esta sección concluye aquí su exploración sobre la investigación y desarrollo de las RNA en el campo del aprendizaje profundo. Aunque no se profundizará más en la historia de las RNA, se deja de forma ilustrativa la Figura 23 con el fin de mostrar los avances significativos que sucedieron después algoritmo de *backpropagation.*

1.3.3 REDES GENERATIVAS ADVERSAS

En 2014, Ian Goodfellow y su equipo, presentaron las redes generativas adversariales/adversas (*Generative Adversarial Network* - GAN). Dicho modelo es un sistema de Aprendizaje Profundo, el cual, es básicamente un enfoque de modelado generativo que crea un nuevo conjunto de datos basado en los datos de entrenamiento. Las GAN tienen dos bloques principales (dos redes neuronales artificiales) que compiten entre sí y son capaces de capturar, copiar y analizar las variaciones de un conjunto de datos. Los dos modelos suelen llamarse Generador y Discriminador. (Agrawal, 2021)

Las dos principales redes neuronales se dividen en dos:

1) Discriminador - Es un simple clasificador que predice si los datos son falsos o reales. Se entrena con datos reales y proporciona información a la red generadora.

2) Generador - La parte generadora de un GAN aprende a crear datos falsos incorporando la retroalimentación del discriminador. Aprende a hacer que el discriminador clasifique su salida como real.

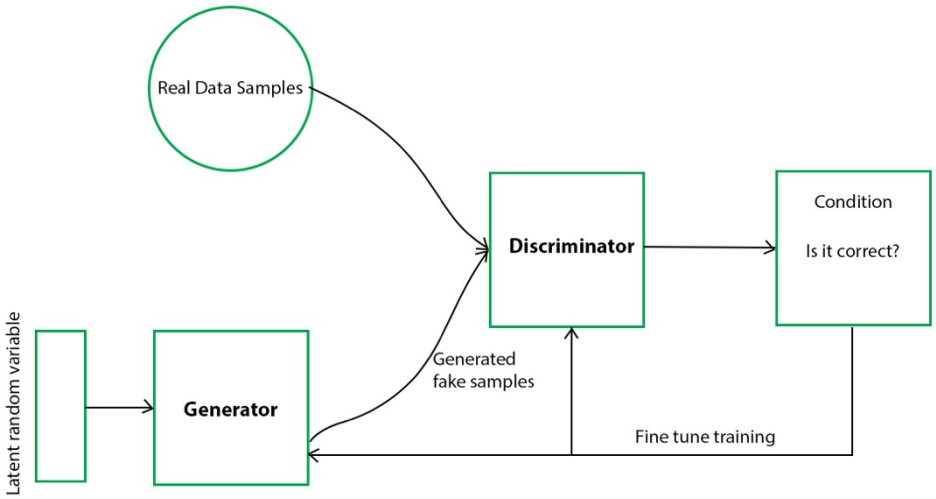


Figura 5. Generative Adversarial Network. Recuperado de: https://www.analyticsvidhya.com/blog/2021/10/an-end-to-end-introduction-to-generative-adversarial-networksgans/

En otras palabras, el generador produce datos nuevos a partir de ruido, mientras que el discriminador distingue entre los datos reales, procedentes del conjunto de datos de entrenamiento, y los datos falsos producidos por el generador. Estas dos redes neuronales -el generador y el discriminador- se entrenan a la par para que compitan entre si hasta que, en determinado punto del entrenamiento de la red GAN, el generador sea capaz de crear datos "realistas" burlando al discriminador, es decir, que no pueda identificar estos datos como falsos. Una vez que esta condición se cumple, el entrenamiento finaliza (véase Figura 5).

Para aprender la distribución P_g del Generador (G) sobre los datos x, se define una prioridad sobre las variables de ruido de entrada $p_z(z)$, luego se representa un mapeo al espacio de datos como $G(z;\theta_g)$, donde G es una función diferencial representada por un perceptrón multicapa con parámetros θ_g. También se define un segundo perceptrón multicapa $D(x;\theta_d)$ que produce un único *ESCALAR*. $D(x)$ representa la probabilidad de que x proceda de los datos y no de P_g. Se entrena el Discriminador D para maximizar la probabilidad de asignar la etiqueta correcta tanto a los ejemplos de entrenamiento como a las muestras de G. Simultáneamente se entrena G para minimizar $\log(1 - D\big(G(z)\big))$. Por lo tanto, El discriminador y el generador juegan un juego Minimax[5] de dos jugadores con una función de valor $V(G,D)$, por lo que la función minimax es:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D\left(G(z)\right))]$$

Dónde:

- $P_{data}(x)$ representa la distribución real de los datos
- P_z representa la distribución del generador
- x representa una muestra de $P_{data}(x)$
- z representa una muestra de P_z
- $D(x)$ representa la salida de la red discriminadora

[5] En teoría de juegos, Minimax es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario y con información perfecta. (Alvarado et al., 2005)

- $G(z)$ representa la salida de la red generadora

Aquí, el Discriminador intenta maximizar el objetivo que es V mientras que el generador intenta minimizarlo, debido a esta minimización/maximización obtenemos el término minimax; G y D aprenden juntos alternando el descenso gradiente (las actualizaciones basadas en el gradiente pueden utilizar cualquier regla de aprendizaje estándar basada en el gradiente).

En otras palabras, el discriminador se actualiza ascendiendo o maximizando su gradiente estocástica:

$$\nabla\theta_d \frac{1}{m}\sum_{i=1}^{m}[\log D\,(x^{(i)} + \log(1 - D\left(G\left(z^{(i)}\right)\right))]$$

Y el generador se actualiza descendiendo o minimizando su gradiente estocástica:

$$\nabla\theta_g \frac{1}{m}\sum_{i=1}^{m}[\log(1 - D(G(z^{(i)})))$$

En la siguiente figura se detalla de forma gráfica la arquitectura general de una red GAN junto con la representación matemática de cada parte que la conforma.

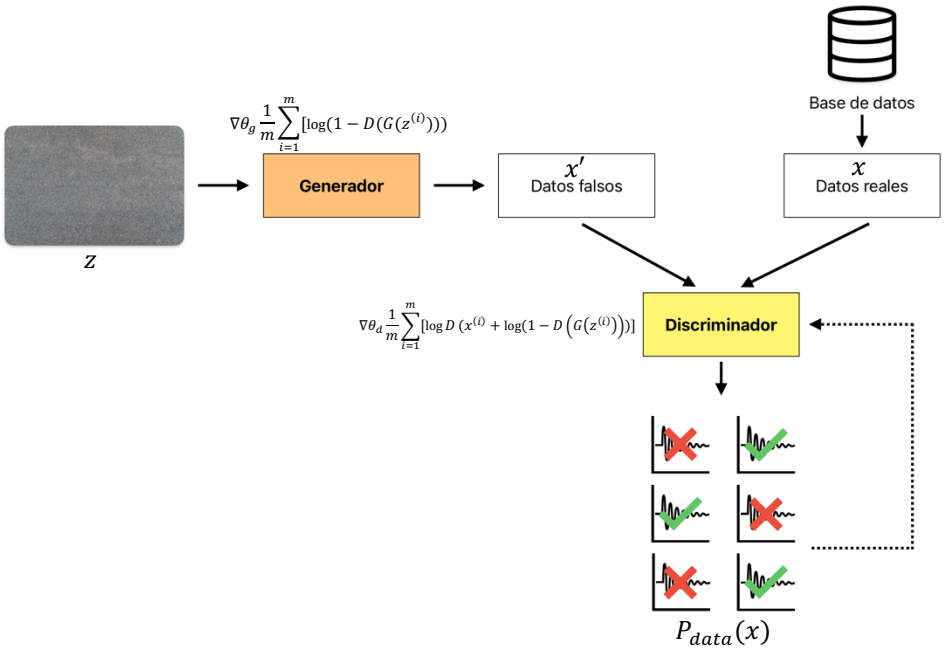


Figura 6. Arquitectura general de una red GAN

Actualmente, las GAN se utilizan bastante para la generación de texto, video, audio y sobre todo, imagen. Por lo que existe una extensa y creciente comunidad relativa al desarrollo de las GAN, por lo cual, si es de interés para el lector saber más de las GAN y estar al tanto de los desarrollos más recientes, se recomienda visitar el siguiente enlace a GitHub:

https://github.com/hindupuravinash/the-gan-zoo

En el, encontrarán un repositorio con una vasta recopilación de distintos proyectos/desarrollos con GAN, de igual forma, con el fin de tener un mayor entendimiento del funcionamiento de este modelo de IA, se recomienda fuertemente leer el artículo *Generative Adversarial Nets* publicado por Ian Goodfellow y su equipo a través de arxiv en el siguiente enlace: https://arxiv.org/abs/1406.2661.

En resumen, el objetivo de este capítulo es dar una noción general de lo que es la inteligencia artificial y sus subcampos, haciendo énfasis en el Aprendizaje Profundo, las redes neuronales artificiales y las redes GAN, ya qué todo este proyecto de investigación se basa en ellos, por lo que es importante entender cómo funciona desde "dentro" este tipo de modelos para tener un mayor entendimiento cuando se detalle la metodología llevada a cabo para el desarrollo del algoritmo propuesto. Por lo que, en el siguiente capítulo se expondrán las tecnologías utilizadas para desarrollar la red tipo GAN, primeras pruebas, los procesos llevados a cabo y el desarrollo e implementación de la red a detalle.

2. IMPLEMENTACIÓN DE REDES GAN PARA LA GENERACIÓN DE SONIDOS

Para el desarrollo de este modelo de IA, se hizo uso del lenguaje de programación Python en su versión 3.10 en conjunto con las librerías TensorFlow y Keras, todo esto desarrollado en el entorno de Google Colab. De igual forma, en un primer momento, se hizo uso del lenguaje de programación *SuperCollider* con el fin de generar una primera base de datos que funja como un *DATASET* de prueba para el primer desarrollo de la red GAN.

Para las primeras pruebas con algoritmos de clasificación de Aprendizaje Profundo, se realizó un script en Python para crear la base de datos, dicho código hace uso de la API de Freesound para la descarga de sonidos mediante búsqueda de texto, etiquetas, autor, año, tipo de sonido, paquetes de sonido, *bitrate*, tipo de licencia, entre otros.

Una vez descargados los sonidos, se aplicaron técnicas de MIR (*Music Information Retrieval*) para extraer los siguientes descriptores: *Spectral Bandwidth*, *Spectral Centroid*, *Spectral Contrast*, *Sprectral Flatness*, *Zero Crossing Rate* y los primeros 20 *MFCC*. Posteriormente, con el *DATASET* de los 6 descriptores mencionados anteriormente, se procedió a la clasificación de los sonidos con un perceptrón multicapa en conjunto con una técnica llamada *EARLY STOP*. Con esto, se logra entrenar el modelo de clasificación manteniendo un error de validación mínimo y, por lo tanto, su precisión sea buena, reduciendo el *OVERFITTING.*

En el algoritmo de clasificación se usaron los datos extraídos (descriptores de audio) como el valor de la variable X y en los valores de la variable Y (*target*) se usaron las etiquetas de los sonidos (9 etiquetas en total). Y, después de numerosas pruebas cambiando parámetros, por ejemplo, en la segmentación de datos, el estado de aleatoriedad, el número de capas ocultas, la función de activación, densidad, número de neuronas artificiales, etc. Se obtuvo como resultado una precisión de 0.25, esto quiere decir un 25% ya que la precisión va de valores de 0 a 1, siendo 1 el 100% de precisión.

A la par, se comenzó a trabajar en otro algoritmo en Python que hace uso de la API de Freesound para descargar sonidos ya con los descriptores de audio seleccionados por el usuario, es decir, cuando se descarga un sonido lo que hace este nuevo algoritmo es que aparte de descargar el archivo de audio, lo descarga con su descriptor o descriptores de audio y estos se guardan en un archivo JSON (véase figura 3). De esta forma se hace más eficiente la creación de la base de datos, tanto de los sonidos como del *DATASET* -en caso de que se utilice la API de Freesound-.

Figura 7. Formato JSON.

Asimismo, se desarrolló otro script para la generación de otra base de datos de sonidos. Este algoritmo fue programado en SuperCollider y el objetivo principal de este código es generar múltiples archivos de audio con la misma frecuencia, duración y envolvente, pero con ligeros cambios en su timbre. (véase las figuras 8- 12)

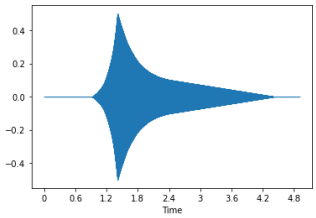


Figura 8. Forma de onda

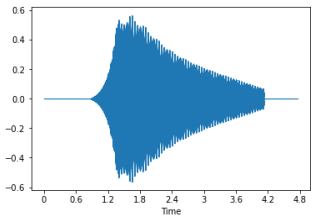


Figura 9. Forma de onda - variación 1

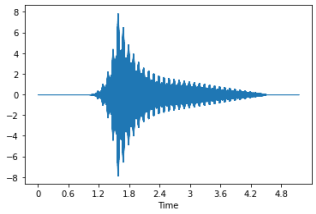


Figura 10. Forma de onda - variación 2

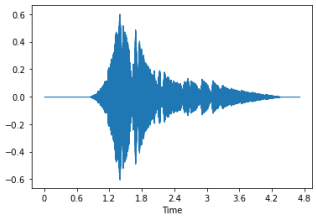


Figura 12. Forma de onda - variación 3

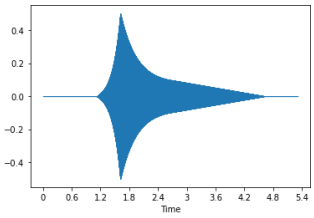


Figura 11. Forma de onda - variación 4

Esto, con el fin de que sea más fácil demostrar que la red neuronal tipo GAN aprendió correctamente de los datos de entrada. En otras palabras, al ser tan similares los sonidos en su envolvente, frecuencia y duración, a la hora de entrenar la red GAN y empezar a generar proceduralemente sonidos a partir de estos, cuando se analicen los sonidos resultantes, se podrá observar de forma clara que, si "aprendió" de forma correcta, los sonidos resultantes tendrían que ser de una envolvente, duración y frecuencia muy parecidos a los reales, si no "aprendió" de forma correcta, los sonidos resultantes serian completamente dispares a los reales, es decir, no tendrían ninguna correlación en su envolvente, duración y frecuencia.

Una vez desarrollado y ejecutado esté algoritmo, se generaron tres mil archivos de sonido con las características mencionadas anteriormente. Posteriormente, con otro script de Python, se usó la librería de procesamiento de audio Librosa[6] en conjunto con la transformada de Hilbert para obtener una representación más limpia de la envolvente de los sonidos generados en SuperCollider.

Una vez extraídos los arreglos de los valores de la envolvente de los primeros tres mil sonidos, se les sacó la media con la librería Numpy y se almacenaron todos los datos en un

[6] https://librosa.org

solo archivo CSV, fungiendo como un primer acercamiento al *DATASET* general con el cual se hará el entrenamiento del primer boceto/prototipo de la red GAN.

Regresando al algoritmo generador de sonidos de Supercollider, se llegó a la idea de que este mismo puede fungir como una aumentador de datos, es decir, usar este mismo algoritmo para la técnica conocida como *DATA AUGMENTATION,* la cual es muy utilizada en aplicaciones de Machine Learning, específicamente en el campo de la visión artificial. Consiste en entrenar una red neuronal, por ejemplo, con cincuenta mil imágenes de caras de perros. Al hacer esto, la red "memorizará" dichas imágenes, por lo que en algún momento se necesitarán nuevas imágenes para obtener mejores resultados. Aquí es donde entra dicha técnica, el *DATA AUGMENTATION* consiste en aplicar filtros, efectos, recortar, redimensionar, deformar, girar horizontalmente y/o verticalmente las fotos para tener más información de las muestras o datos que ya existen, pasando de tener una base de datos de cincuenta mil archivos a cien mil o doscientos mil datos, dependiendo de que tantas transformaciones se apliquen a la base de datos original. Con esto, se obtiene un modelo con más datos de entrada, un mejor entrenamiento, mejores resultados y más importante aún, se evita el *OVERFITTING*.

Siguiendo con esta idea del *DATA AUGMENTATION*, se puede aplicar esta misma técnica al sonido. Si la base de datos inicial cuenta con tres mil sonidos, aplicando filtros, efectos y diferentes transformaciones a los audios originales, se puede conseguir triplicar o cuadruplicar fácilmente -incluso más- la base de datos. Esto con el propósito de entrenar con más datos la red GAN, obtener mejores resultados y una conseguir una precisión óptima en el funcionamiento del modelo.

2.1 DESARROLLO DE RED GAN

En su primera versión, la red GAN fue pensada para trabajar con espectrogramas de la base de datos de sonidos generada con SuperCollider. Por lo que, se desarrolló un script en Python que mostrara la imagen del espectro de los sonidos, usando la librería Librosa. (Figura 13)

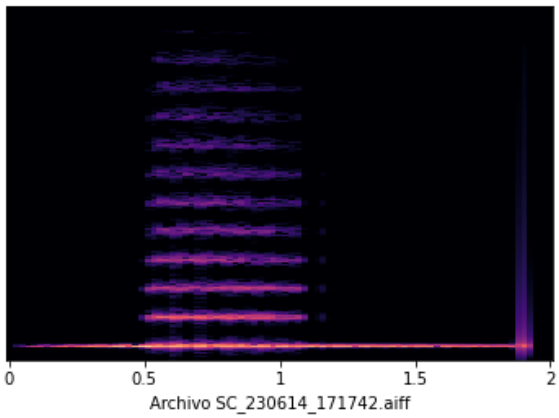


Figura 13. Espectrograma

Una vez teniendo los espectrogramas de todos los sonidos -3,000- se segmentaron en dos grandes categorías:

- ⇒ Train – Para entrenar a la red GAN con un pequeño porcentaje de las imágenes totales divididos en 2 clases, Class_1 y Class_2.
- ⇒ Validation – Para validar las imágenes "reales" contra las imágenes del entrenamiento -Train- y obtener la matriz de los datos para empezar a generar las imágenes. (Figura 14)

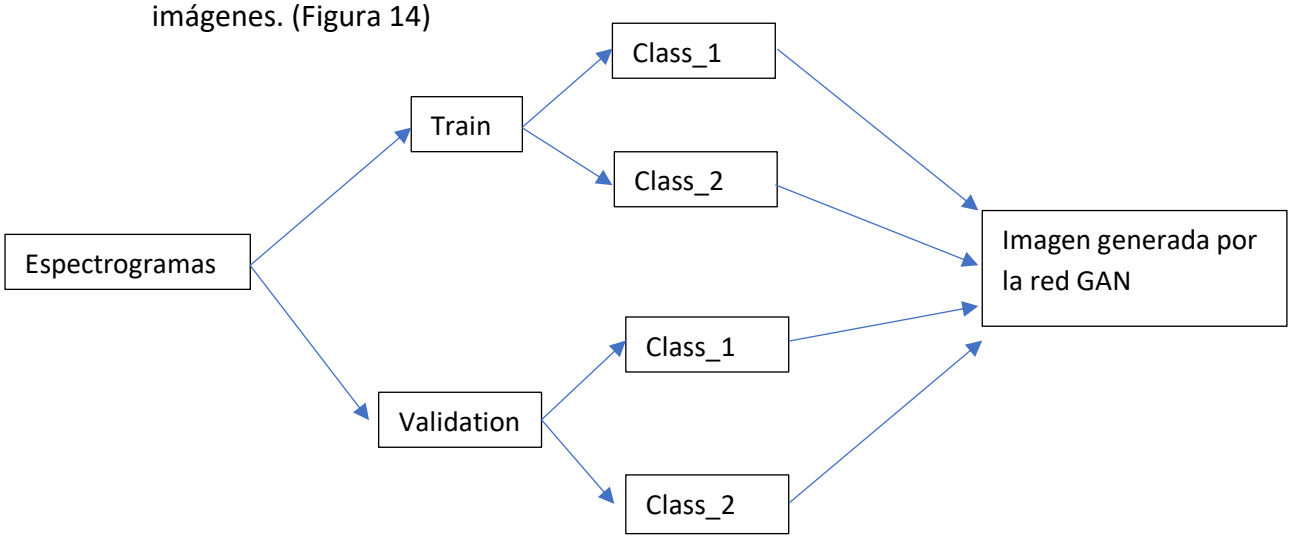


Figura 14. Entrenamiento de red GAN - Prueba 1

A grandes rasgos, las imágenes se cargaron en los parámetros de la red GAN para ser preprocesados; esto quiere decir que, se hizo una redimensión de los pixeles, en este caso (288px, 432px) y aleatorizar las imágenes para tener un preprocesamiento más rápido. Después, se normalizaron los datos de valores RGB a escalas de grises para optimizar aún más los datos de las imágenes. Siguiendo con esta lógica, al usar datos con valores de 288px * 432px se necesitan al menos 124,416 neuronas en total para que la red GAN pueda procesar todo el *DATASET*.

Con 1,000 épocas y un *BATCH SIZE* pequeño -20- se compiló la red GAN con su discriminador y su generador para empezar a procesar los datos y generar proceduralmente las imágenes, partiendo de ruido. (Figura 13)

Figura 17. Ruido creado por el generador – época 0

Posteriormente, al pasar las 1000 épocas, se generaron dos imágenes de dos espectrogramas (Figura 14), las cuáles son el resultado final del entrenamiento de la red GAN. Dichas imágenes sirven para comprobar que la red si está creando las imágenes deseadas basadas en las imágenes reales de entrada.

Figura 18. Imagen de espectrograma creado por el generador – época 1000

Una vez comprobado el funcionamiento correcto de la red, se procedió a trabajar con audio directamente, en lugar de ocupar las imágenes de los espectrogramas, se decidió usar el mismo espectrograma pero como *ARRAY* de valores, es decir, analizar cada audio de la base de datos, obtener los valores del espectrograma con funciones propias de TensorFlow (TF) y posteriormente almacenarlos en un tensor.

El código que se desarrolló para generar sonido se encuentra en el anexo, en el primer apartado: *CÓDIGO DE RED GAN*. Dicho esto, en la siguiente sección se expondrá la arquitectura de la red generadora y discriminadora, siguiendo por una breve descripción del proceso de entrenamiento de la misma red y las decisiones que se llevaron a cabo para poder implementar una interfaz y la opción de poder procesar los audios en crudo generados por la GAN para transformarlos con múltiples efectos y, finalmente, se expondrá el entorno en Google Colab dónde se encuentra la red entrenada lista para generar sonido y las implementaciones mencionadas anteriormente.

2.1.1 RED GENERADORA

Para la arquitectura de la red generadora, se utilizó el modelo *Sequential* de TF, heredada del método *Model,* la cuál es un modelo que agrupa capas en un objeto con características de entrenamiento/inferencia, específicamente al usar *Sequential*, las capas se agrupan de forma lineal.

Las capas utilizadas para esta red fueron:

- *Dense:* Se refiere a una red neuronal regular densamente conectada (véase la figura 15 en el anexo).
- *BatchNormalization*: Esta capa normaliza las entradas. Aplica una transformación que mantiene la media de salida cercana a 0 y la desviación estándar de salida cercana a 1.
- *Reshape*: Reconfigura las entradas en la forma definida por el usuario.

Código de la red generadora de la GAN:

```
generator = tf.keras.Sequential([
    layers.Dense(256, input_shape = (latent_dim,), activation =
'relu'),
    layers.BatchNormalization(),
    layers.Dense(512, activation = 'relu'),
    layers.BatchNormalization(),
    #layers.Dense(784, activation = 'tanh'),
    layers.Dense(337 * 513, activation = 'tanh'),
    layers.Reshape((337, 513, 1))
])
```

Los *HIPERPARÁMETROS* definidos en el código fueron variando sus valores hasta ajustarse al modelo y lograr una mayor precisión.

El sumario que da como resultado de la arquitectura de la red generadora es la siguiente:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               25856

 batch_normalization (Batch  (None, 256)               1024
 Normalization)

 dense_1 (Dense)             (None, 512)               131584

 batch_normalization_1 (Bat  (None, 512)               2048
 chNormalization)

 dense_2 (Dense)             (None, 60534)             31053942

 reshape (Reshape)           (None, 118, 513, 1)       0

=================================================================
Total params: 31214454 (119.07 MB)
Trainable params: 31212918 (119.07 MB)
Non-trainable params: 1536 (6.00 KB)
```

2.1.2 RED DISCRIMINADORA

Igualmente, para la red discriminadora se utilizó el modelo *Sequential* de TF.

Las capas para esta red son:

- *Dense*. Capas mencionadas anteriormente
- *Flatten.* Estas capas solo aplanan la entrada. No afecta al tamaño del lote (*BATCH SIZE*).

Código de la red discriminadora de la GAN:

```
discriminator = tf.keras.Sequential([
    layers.Flatten(input_shape = (337, 513, 1)),
    layers.Dense(512, activation = 'relu'),
    layers.Dense(256, activation = 'relu'),
    layers.Dense(1, activation = 'sigmoid')
])

discriminator.compile(optimizer =
tf.keras.optimizers.Adam(learning_rate = 0.0002),
                      loss = tf.keras.losses.BinaryCrossentropy(),
```

Esta red se compiló usando un optimizador que implementa el algoritmo *Adam*[7] y una clase de perdida de tipo *tf.keras.losses.BinaryCrossentropy* para calcular la pérdida de entropía cruzada entre las etiquetas verdaderas y las etiquetas predichas.

El sumario que da como resultado de la arquitectura de la red discriminadora es la siguiente:

[7] La optimización *Adam* es un método estocástico de descenso de gradiente que se basa en la estimación adaptativa de momentos de primer y segundo orden. (López, 2023)

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 60534)             0

 dense_3 (Dense)             (None, 512)               30993920

 dense_4 (Dense)             (None, 256)               131328

 dense_5 (Dense)             (None, 1)                 257

=================================================================
Total params: 31,125,505
Trainable params: 31,125,505
Non-trainable params: 0
```

2.1.3 ENTRENAMIENTO DE RED GAN

Para el entrenamiento de la red, se definió el número de épocas a 1,000, dónde cada 100 épocas se guarda un sonido generado por la red durante el entrenamiento, con el fin de obtener un resultado sonoro para comprobar el funcionamiento del entrenamiento.

En el siguiente enlace se encuentra una carpeta con 10 ejemplos de sonidos generados cada 100 épocas durante en el entrenamiento de la red GAN. Dicho entrenamiento se realizó haciendo uso de un *DATASET* abierto del proyecto *Magenta*, específicamente un *DATASET* con 4,096 sonidos de *Nsynth*[8].

https://drive.google.com/drive/folders/1cJAINOE8BNLGBemQUpNnrCHkeeNp7kKa?usp=share_link

Dentro de la carpeta, se pueden escuchar los audios generados, en esta primera etapa es bastante recomendable subir el volumen del dispositivo donde esté abriendo los archivos ya qué tienen muy poca ganancia. Como ejemplo de cómo funciona la red GAN, se tomará en cuenta el primer archivo de audio generado llamado *0sc.wav*, dicho archivo se generó en la época 1, cómo se puede escuchar, es ruido, como se ejemplifica en la Figura 6. Escuchando los demás sonidos de la carpeta, se puede notar que la red generadora va mejorando por cada época que pasa del entrenamiento, a pesar de que los audios tengan un ligero cambio en su timbre, se nota una pequeña mejoría entre ellos.

Es por esto por lo que se tomó la decisión de implementar un método de *Spectral Gating,* el cuál es una forma de *Noise Gate* o "puerta de ruido" que, a grandes rasgos, reduce el ruido en señales de tiempo. Dicho método se aplicó mediante la librería de Python *noisereduce*[9].

Una vez que el modelo de red GAN se entrenó de forma exitosa y se aplicó el método de *Spectral Gating* para mejorar la calidad de los sonidos, la red se guardó en un *ARCHIVO .H5*, el cual contiene los valores óptimos de ponderaciones, sesgos y, por ende, el modelo entrenado. Este archivo se usa porque permite almacenar una gran cantidad de datos en forma de matrices multidimensionales y la recuperación y análisis de los datos es sumamente rápida. Por lo que no hay qué volver a entrenar el modelo, solo cargar el

[8] https://magenta.tensorflow.org/datasets/nsynth#files
[9] https://pypi.org/project/noisereduce/

modelo con el *ARCHIVO .H5,* generar una semilla aleatoria entre 1 y el valor definido en el hiperparámetro del espacio latente o dimensión latente; finalmente, se genera un audio de 2 segundos con la función *predict* y se guarda el archivo wav a 44100 KHz.

Siguiendo con la idea principal de que este desarrollo funja para la composición asistida de música electroacústica (al menos en este primer desarrollo/implementación) se buscó que no solo sea para generar sonido en crudo, sino qué se puedan generar variaciones del mismo mediante distintas técnicas de síntesis de sonido, al igual que todo lo demás, implementado en Python. Es por esto por lo que se recurrió a la librería de Spotify *Pedalboard*[10]*,* con el fin de leer, renderizar y aplicar distintos efectos a los sonidos en crudo generados por la GAN. Los efectos seleccionados se dividen en 4 categorías:

1. Efectos de sonoridad y rango dinámico*: Compressor, Gain, Limiter.*
2. Ecualizadores y filtros: *HighpassFilter, LadderFilter, LowpassFilter.*
3. Efectos espaciales: *Delay, Reverb.*
4. Efectos varios: *Chorus, Distortion, Phaser, Clipping.*

Por otro lado, otra de las ideas principales de este desarrollo es que cualquier usuario pueda usarlo sin la necesidad de saber de programación o de IA. Es gracias a esto, qué se decidió desarrollar una interfaz para optimizar los pasos de generación de audio y, en caso de que el usuario quiera, aplicar los efectos mencionados anteriormente. Para esta última fase, se hizo uso de la librería *ipywidgets*[11] para implementar *sliders*, botones, cajas, pestañas entre otros.

En la siguiente y última sección de este capítulo se hablará sobre la interfaz, los widgets y el funcionamiento general del modelo, así como una breve descripción de como generar sonidos y como aplicar efectos a partir de la interfaz y de la red ya entrenada.

[10] https://spotify.github.io/pedalboard/index.html
[11] https://ipywidgets.readthedocs.io/en/stable/

2.1.4 SISTEMA DE GENERACIÓN DE GESTOS SONOROS PARA LA COMPOSICIÓN ASISTIDA CON INTELIGENCIA ARTFICIAL

En el siguiente enlace se puede encontrar la página de Google Colab con el código y la interfaz finalizada:

https://colab.research.google.com/drive/1gMpWnwGAEddZNFv6sMBH529x95AMny0i?usp=sharing

Cómo se puede observar, es sumamente sencilla la interfaz (Figura 20) y bastante intuitiva ya que consta de solo 3 grandes “pasos”:

1. Importar librerías.

   En este primer apartado y en todos los demás, hay que dar clic en el símbolo  para ejecutar la celda y automáticamente se descargarán todas las librerías necesarias para poder ejecutar las demás celdas.

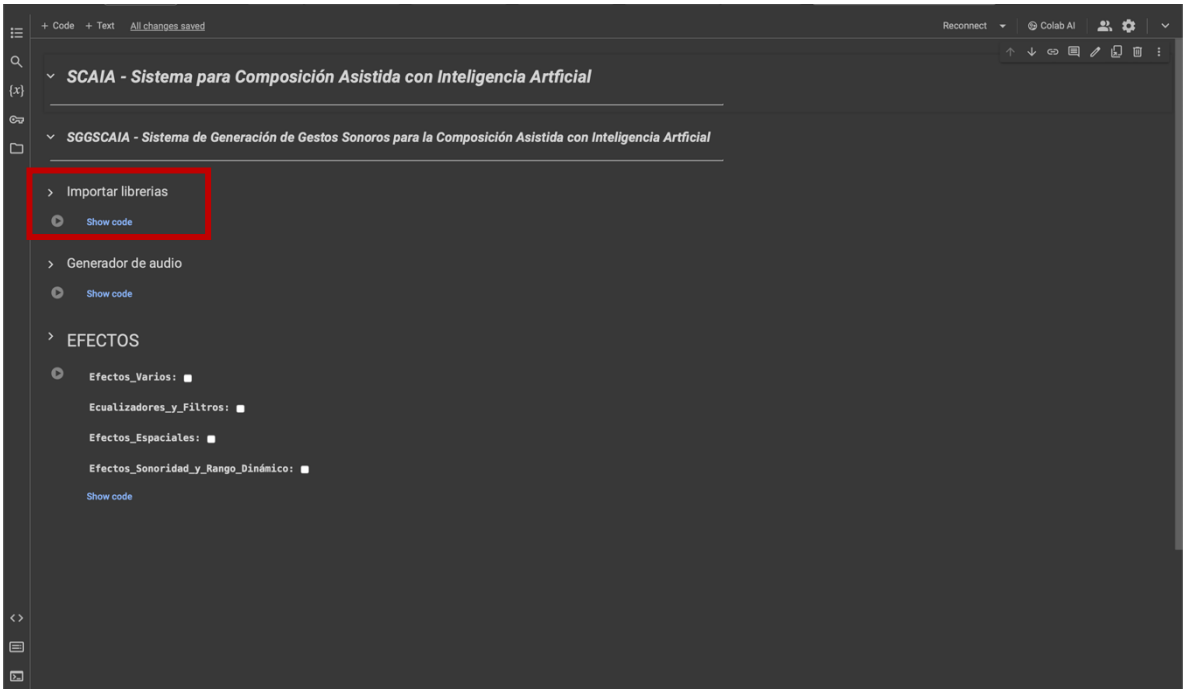


Figura 19. Interfaz principal

2. Generador de audio.

   Después de dar clic en Ejecutar celda, aparece un botón con la descripción “Generar audio” esto significa el código ya se compiló y está listo para generar sonido; al darle clic a dicho botón, después de ~ 8 segundos, se generará un sonido en crudo con el método de *Spectral Gating* ya implementado y se reproducirá automáticamente, al mismo tiempo, se desplegará una visualización de la forma de onda del sonido generado por la red GAN.

   Esta celda puede ejecutarse cuantas veces quiera el usuario para generar varios sonidos, el único inconveniente es que los sonidos generados son muy parecidos entre sí.

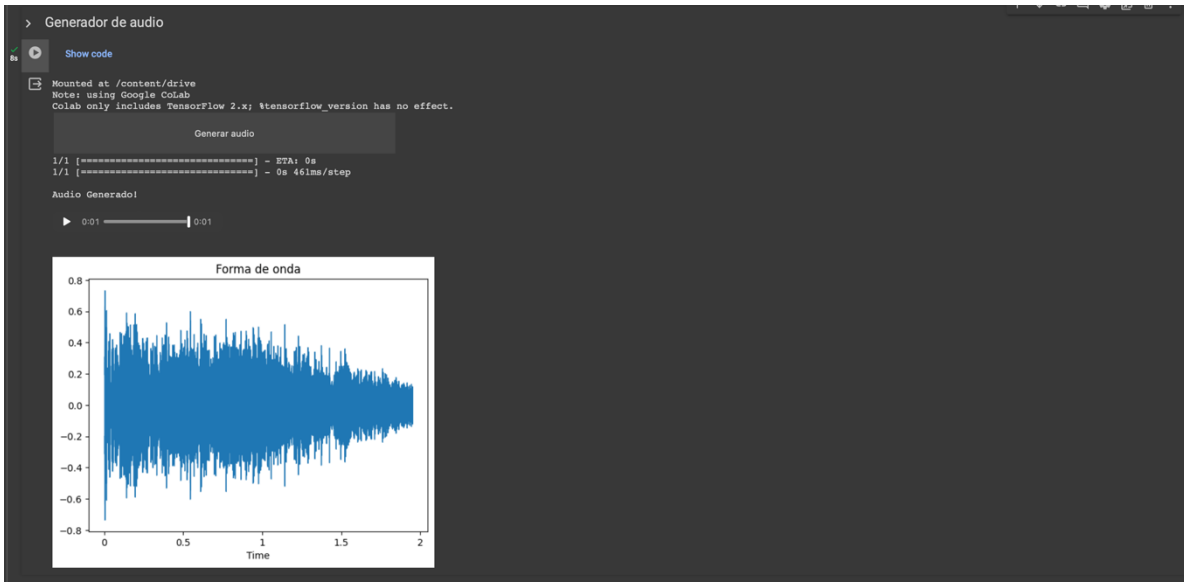


Figura 20. Interfaz principal - Generador de audio

3. EFECTOS.

   Para este último apartado, al igual que las celdas anteriores, después de dar clic en Ejecutar celda, aparecerán los grupos de efectos mencionados en la página 36, dependiendo de los efectos que el usuario seleccione en el apartado A (Figura 22) saldrá una leyenda donde se muestre si están activados los filtros seleccionados, en caso de que no aparezcan como activos los efectos seleccionados, solo se da clic en Ejecutar celda y se resuelve el problema.

Posterior a eso, en el apartado B (Figura 22) el usuario podrá controlar a su gusto los parámetros de cada efecto seleccionado.

Igual que en el apartado anterior, después de haber seleccionado los efectos deseados y modificar sus parámetros, se le hace clic al botón que aparece en la parte inferior con la descripción “Aplicar transformaciones”. En este punto, el botón solo devuelve el audio transformado, no despliega visualización de la forma de onda como en el apartado anterior.

Si el usuario quiere agregar o desactivar efectos, tendrá que volver a ejecutar la celda para qué el código se actualice con los efectos seleccionados o deseleccionados en su defecto.

Figura 21. Interfaz principal - EFECTOS

3. EXPLORANDO LA GENERACIÓN PROCEDURAL DE AUDIO EN LA COMPOSICIÓN DE MÚSICA ELECTROACÚSTICA

3.1 COMPOSICIÓN DE LA OBRA 'LEVIATHAN'

Desde mis primeras incursiones en la composición de música electroacústica hasta mi evolución actual, he buscado constantemente nuevas formas para acelerar el flujo de composición, específicamente, en la búsqueda de sonidos. Esto dio paso a explorar el campo de la IA con el objetivo de desarrollar un "asistente" de composición de música electroacústica, lo qué me llevó a desarrollar este proyecto, con el fin de obtener un modelo que sirva como generador de gestos sonoros[12] para aplicarlos en mis composiciones. A lo largo de este capítulo, expondré el uso de este modelo GAN en la composición de mi último trabajo de electroacústica y la función de la red dentro de la misma.

En mi proceso creativo, primero me dedico a la confección de maquetas sonoras mediante la grabación y modificación de distintos sonidos. Este proceso implica una continua exploración de posibilidades auditivas, con la constante búsqueda de nuevas incorporaciones sonoras. En el contexto específico de mi obra Leviathan[13], he recurrido a la asistencia de la red GAN mencionada con anterioridad, la cual, fungió como una herramienta complementaria en mi proceso de composición.

Para empezar, voy a diseccionar por partes mi obra, donde cada parte representa un punto donde se usó algún gesto generado por la red GAN. Cabe desatacar que la gran mayoría de los gestos generados por la red se usaron para acentuar puntos donde hay una subida en la energía de la obra.

[12] Desde el punto de vista de Dennis Smalley, un gesto sonoro se puede entender como "una trayectoria de energía-movimiento que excita el cuerpo sonoro" (Smalley, 1997, p. 111).
[13] https://on.soundcloud.com/s6NJ2

⇒ Minuto 1:58 / Gesto-1

Este primer gesto, lo utilicé para darle más brillo al gesto que suena en el minuto 1:59; gracias a que el gesto-1 tiene un mayor registro en las frecuencias altas y que usé un ecualizador con un *low cut*, me permitió hacer un balance con los otros dos sonidos que van desde las frecuencias bajas hasta las medias, lo que da como resultado una suerte de gesto de impulsión mantenido aprovechando la resonancia propia del gesto-1 para poder finalizar o terminar orgánicamente la subida de energía en este punto combinando la caída o *release* del gesto general con la textura de fondo.

⇒ Minuto 2:19 / Gesto-2

En este momento, al haber una gran variedad de sonidos ocurriendo al mismo tiempo, se utilizó el gesto-2 para darle énfasis a un "golpe" de otro sonido.

El gesto-2, al ser un sonido "eléctrico", diría yo, por su sonido brillante y chispeante, se incorpora muy bien con los demás sonidos de fondo porqué, tímbricamente, son muy parecidos, y el contexto sonoro en este punto donde hay mucho movimiento y en cierta forma es un punto de tensión que da paso a un momento de calma, este gesto-2 logra ser un sonido *ad hoc* al contexto que ocurre en este punto.

Este gesto-2 fue usado en crudo, es decir, así como se generó así se usó, sin ningún tipo de procesamiento.

⇒ Minuto 3:14 / Gesto-3

Al igual que el gesto anterior, el gesto-3 se usó en crudo, aprovechando sus cualidades tímbricas para resaltar y terminar el gesto general que ocurre en este momento. De igual forma, el *release* del gesto-3 se combina con una textura que tiene similitud en las frecuencias altas.

⇒ Minuto 4:00 / Gesto-3

Para este momento, se usó el mismo gesto-3 con la misma finalidad que en el minuto 3:14, la única diferencia es que, la obra llega a un punto caótico dónde la

función principal del gesto-3, es simplemente potenciar otro golpe sostenido que se va transformando en el tiempo durante un par de segundos.

⇒ Minuto 4:54 / Gesto-4

Aquí, el gesto-4 se transformó con un *delay,* un *stutter* y se le bajó el tempo. En este punto, es dónde sucede el segundo momento más caótico de la obra. El gesto-4 se mezcla con los demás sonidos para dar paso a un movimiento donde todos los sonidos convergen, dando una sensación de "transformación" energética que presenta un breve momento donde todos los gestos suenan sin ninguna textura de fondo y se transforman constantemente.

⇒ Minuto 5:22 / Gesto-5 y 6

En este momento, hay una subida breve en la energía del sonido con el golpe que suena aproximadamente en el minuto 5:23, lo cual, es resultado de la combinación de un gesto x con las dos modificaciones del gesto-2.

Por un lado, el gesto-5 es una variación con *reverb,* al cual se le bajaron 8 semitonos e igualmente, se bajó un poco la ganancia de este gesto. Para el gesto-6, se usó un ecualizador para cortar las frecuencias graves y agudas, haciendo un ligero levantamiento en las frecuencias medias, también se le bajaron 7 semitonos y un poco a la ganancia. De esta forma, logré casi igualar el timbre del gesto x, el cuál suena como un golpe seco, logrando qué, perceptualmente, el gesto-5 y 6 suenen como consecuencia del golpe en seco. Finalmente, para ligar y lograr una mayor cohesión con los sonidos consecuentes, jugué con los valores de la reverberación y automaticé el volumen para que vaya decreciendo, dando como resultado una sensación de que este gesto se transformó en una textura casi similar que queda sonando un par de segundos y, a su vez, esta textura da paso al siguiente gesto.

⇒ Minuto 5:38 / Gesto-2

Por último, volví a utilizar el gesto-2, esta vez 7 semitonos abajo, para darle énfasis y resonancia al gesto que suena en el minuto 5:39 aproximadamente. Siendo este, el último gesto que utilicé en toda la obra de Leviathan.

Cabe mencionar que, la conceptualización de esta obra la comencé alrededor de octubre del 2023 y empecé a componer a mitades del mes de noviembre del mismo año. Por lo que la implementación de los gestos sonoros creados por la red GAN seguían en prueba, aunque he de admitir personalmente, que al escuchar estos resultados generados me agradaron bastante.

Por otro lado, al momento de la composición de Leviathan, la red GAN seguía en desarrollo.

Actualmente, estoy trabajando en una composición de música electroacústica de 1 minuto para una convocatoria en Brasil, por lo que estos meses que me quedan de la maestría voy a seguir optimizando y mejorando la red GAN, hasta donde mis capacidades técnicas puedan. Esto, con el fin de poder crear dicha composición con una mayor variedad de sonidos generados por la red.

Por otro lado, sigo buscando cómo mejorar la calidad de los sonidos generados y cómo hacer más eficiente y optimizada la generación de estos.

4. CONCLUSIONES

El empleo de la red GAN para la generación de gestos sonoros aplicados a la composición de música electroacústica presenta numerosos beneficios. En primer lugar, la GAN ofrece una amplia gama de posibilidades creativas al proporcionar una fuente casi inagotable de nuevos sonidos que pueden enriquecer y diversificar la paleta sonora del compositor. Además, la capacidad de la GAN para capturar patrones complejos en los datos de entrenamiento puede permitir la creación de gestos sonoros altamente detallados y complejos, brindando una optimización en el flujo del proceso creativo/sonoro.

Sin embargo, es importante reconocer la complejidad inherente al desarrollo de una red GAN, que requiere de una cuidadosa selección de arquitecturas y parámetros, así como un extenso proceso de entrenamiento y ajuste. Este proceso puede ser bastante laborioso, demandando recursos computacionales significativos y expertise en el campo del aprendizaje automático. Además, los resultados generados por la GAN pueden no siempre ser favorables y pueden mostrar tendencias a la repetición o producir artefactos no deseados. Esto puede requerir de un análisis crítico por parte del compositor para discernir qué resultados son útiles y cuáles necesitan ser descartados o refinados.

En cuanto al futuro, se presenta un continuo trabajo para optimizar y mejorar la red GAN existente. Esto puede incluir la exploración de nuevas arquitecturas, técnicas de entrenamiento más avanzadas y la incorporación de datos de entrenamiento más diversos y representativos. Al hacerlo, se espera una mayor originalidad, “expresión”, mejoría en la calidad y en la diversidad de los gestos sonoros generados, ofreciendo mejores resultados para ser implementados en la composición de música electroacústica.

Actualmente, sigo trabajando en la implementación de otra red generadora y discriminadora para evaluar cuáles funcionan mejor en la generación de gestos sonoros.

5. REFERENCIAS

1. *Escobar Castañeda, A. A. (2018). Hacia una escucha automática de la espontaneidad: relaciones complejas entre la libre improvisación, la escucha y los sistemas de aprendizaje automático. [Tesis de maestría]. UNAM.*

2. *Castro Solano, O. A. (2022). Estrategias de composición a través de la similitud tímbrica mediante la sinergia entre la escucha del compositor y la escucha de máquina. [Tesis de doctorado]. UNAM.*

3. Iberdrola. (s. f.). *Machine Learning*. Recuperado el 5 de diciembre de 2022, de https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico

4. Stewart, M. (2021, 9 diciembre). *Comprehensive Introduction to Autoencoders - Towards Data Science*. Medium. https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

5. Kingma, D. P. (2016, 15 junio). *Improving Variational Inference with Inverse Autoregressive Flow.* ArXiv.Org. https://arxiv.org/abs/1606.04934

6. IBM. (2021, 7 abril). *Recurrent Neural Networks*. https://www.ibm.com/cloud/learn/recurrent-neural-networks

7. Collins, N. (2011). *Autocousmatic: automatic electroacoustic music*. Nick Collins. https://composerprogrammer.com/autocousmatic.html

8. Frenzel, M. (2019, 19 diciembre). SampleVAE -A Multi-Purpose AI Tool for Music Producers and Sound Designers. Max Frenzel. https://maxfrenzel.com/articles/samplevae-a-multi-purpose-ai-tool-for-music-producers-and-sound-designers

9. Artificial Intelligence (AI). (2020, 3 junio). IBM. https://www.ibm.com/cloud/learn/what-is-artificial-intelligence#toc-what-is-ar-DhYPPT4m

10. McCarthy, J. (2007, 12 noviembre). WHAT IS ARTIFICIAL INTELLIGENCE? Formal Reasoning Group Stanford. http://www.formal.stanford.edu/jmc/whatisai/whatisai.html

11. Agrawal, R. (2021, 12 noviembre). *An End-to-End Introduction to Generative Adversarial Networks (GANs)*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/an-end-to-end-introduction-to-generative-adversarial-networksgans/

12. Casas Roma, J., Lozano Bagén, T., & Bosch Rué, A. (2020). *Deep learning: Principios y fundamentos*. Universitat Oberta de Catalunya. https://www-digitaliapublishing-com.pbidi.unam.mx:2443/a/69970

13. Donahue, C., McAuley, J., & Puckette, M. (2018). Adversarial Audio Synthesis. *ArXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1802.04208

14. Casas Roma, J., Lozano Bagén, T., & Bosch Rué, A. (2020). Deep learning: Principios y fundamentos. Universitat Oberta de Catalunya. https://www-digitaliapublishing-com.pbidi.unam.mx:2443/a/69970

15. Torra Porras, S. (2004). *Siniestralidad en seguros de consumo anual de las entidades de previsión social, La. Perspectiva probabilística y econométrica. Propuesta de un modelo econométrico neuronal para Cataluña*. (Tesis de Doctorado). Universitat de Barcelona. http://hdl.handle.net/10803/1455

16. Greer, K. (2012). *Turing: Then, Now and Still Key,* En Yang, X. (Ed.). *Artificial Intelligence, Evolutionary Computing and Metaheuristics*. Springer. DOI 10.1007/978-3-642-29694-9

17. IBM Documentation. (s.f.). El Modelo de Redes Neuronales. Recuperado el 15 abril de 2023 . https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model

18. What is overfitting?. (s. f.). Amazon Web Services, Inc. Recuperaod el 1 de junio de 2023. https://aws.amazon.com/es/what-is/overfitting/#:~:text=Overfitting%20is%20an%20undesirable%20machine,on%20a%20known%20data%20set.

19. IBM documentation. (2021, 17 agosto). https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model

20. Torres, A. (2017, 17 agosto). Ley de Hebb: la base neuropsicológica del aprendizaje. *Psicología y Mente*. https://psicologiaymente.com/neurociencias/ley-de-hebb

21. Eiriz, C. (2019). Una guía comentada acerca de la tipología y la morfología de Pierre Schaeffer. *Cuadernos del Centro de Estudios en Diseño y Comunicación*, *39*. https://doi.org/10.18682/cdc.vi39.1757

22. Schaeffer, P. (1996). *Tratado de los objetos musicales*. Alianza Editorial Sa.

23. Chion, M. (1983). *Guide des objets sonores: Pierre Schaeffer et la recherche musicale*. Paris: Buchet/Castel.

24. Alvarado, J., García, J., Hernández, W., Pimentel, N., & Tristán, M. (2005, 13 junio). Minimax. Recuperado 5 de marzo de 2024, de https://nlaredo.tecnm.mx/takeyas/Apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/Minimax/Minimax.pdf

25. Areli-Toral, J. (s. f.). Redes neuronales. Recuperado 6 de enero de 2024, de http://www.cucei.udg.mx/sites/default/files/pdf/toral_barrera_jamie_areli.pdf

26. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems.

27. Archivos H5. (s. f.). FileExt. Recuperado 6 de enero de 2024, de https://filext.com/es/extension-de-archivo/H5#google_vignette

28. S. Matos, J. (s. f.). Magnitudes Físicas. Recuperado 6 de enero de 2024, de https://www2.ulpgc.es/hege/almacen/download/35/35543/tvectores.pdf

29. De los Santos, P. R. (2023, 29 junio). Deep Learning para todos los públicos: ¿Qué son los tensores? ¿Qué es TensorFlow? Telefónica Tech. https://telefonicatech.com/blog/deep-learning-para-todos-los-publicos

30. Galán, R., Jiménez, A., Sanz, R., & Matía, F. (2000). Control Inteligente. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, 4(10), 43-48.

31. Redes Neuronales Artificiales y sus Aplicaciones. (2008). En Basogain, X. (Comp.), *Redes Neuronales Artificiales y sus Aplicaciones* (pp. 2). Escuela Superior de Ingeniería Bilbao UPV-EHU.

32. López, C. C. (2023). Algoritmo de redes neuronales como herramienta de asesoramiento en la ciberatribución. Ciencia y Tecnología, 31-48. https://doi.org/10.18682/cyt.vi23.9969

33. Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017.

6. GLOSARIO DE TÉRMINOS

6.1 ARCHIVO .H5

La extensión de archivo H5 es un formato de archivo de código abierto desarrollado por el Centro Nacional de Aplicaciones de Supercomputación (NCSA). Es un tipo de archivo de biblioteca utilizado para almacenar grandes cantidades de datos numéricos, gráficos y de texto. Como su nombre indica, la característica peculiar del formato de archivo H5 es que los datos se organizan de forma jerárquica y, por tanto, puede gestionar una gran cantidad de datos de forma muy eficiente. (Archivos H5, recuperado el 6 de enero de 2024)

6.2 ARRAY

En programación, un *array* o arreglo es un tipo de dato estructurado que permite almacenar y acceder a un conjunto de elementos del mismo tipo en una ubicación de memoria contigua.

6.3 BATCH SIZE

Es el número de ejemplos que se introducen en cada iteración de la red neuronal para que se entrene.

6.4 DATA AUGMENTATION

Data Augmentation o Aumento de Datos es el proceso de generar y/o modificar nuevos datos a partir de los datos reales. Principalmente se usa para diversificar e incrementar el tamaño del *dataset* de entrenamiento.

6.5 DATASET

Es un conjunto de datos, una colección estructurada de información que, generalmente, se utiliza para respaldar bases de datos, analizar patrones, entrenar modelos de IA, entre otros.

6.6 EARLY STOP

Consta de detener el entrenamiento en el momento que se observe un incremento en el valor del error de validación.

6.7 ESCALAR

Una magnitud física se denomina escalar cuando queda completamente caracterizada mediante un número real (cuyo valor es independiente de cualquier sistema de ejes) y la unidad de la magnitud (sistema de unidades fijado). (S. Matos, recuperado el 6 de enero de 2024)

6.8 HIPERPARÁMETROS

Son variables externas que no se obtienen a partir de los datos, se definen de forma manual antes de entrenar un modelo.

6.9 OVERFITTING

El *overfitting* o sobreajuste es un comportamiento de aprendizaje automático no deseado que se produce cuando el modelo de aprendizaje automático proporciona predicciones precisas para los datos de entrenamiento, pero no para los datos nuevos. (Amazon, recuperado el 1 de junio de 2023)

6.10 TENSORES

Los tensores son objetos matemáticos que almacenan valores numéricos y que pueden tener distintas dimensiones. (De los Santos, 2023)

7. ANEXO

7.1 CÓDIGO DE RED GAN

El siguiente código se realizó usando Google Colab, por lo que hay rutas de acceso hacia Google Drive para acceder a sonidos que están almacenados en la nube; se recomienda usar algún entorno de Colab para ejecutar el código completo o bien, si se quiere usar el código de forma local, en su máquina, solo se necesita cambiar las rutas de acceso de los sonidos.

Primero se necesita importar las siguientes librerías:

```
pip install pedalboard
pip install noisereduce
pip install pydub
pip install matplotlib
pip install tensorflow==2.15.0
pip install soundfile
pip install librosa
pip install numpy
pip install ipython
```

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import models
import librosa
import os
from os import listdir
from os.path import isfile, join
import soundfile as sf
plt.style.use('seaborn-whitegrid')

try:
    from google.colab import drive
    drive.mount('/content/drive', force_remount=True)
    COLAB = True
    print("Note: using Google CoLab")
    %tensorflow_version 2.x
except:
    print("Note: not using Google CoLab")
    COLAB = False

tf.random.set_seed(42)
```

```
frame_length =  1024 #2048
frame_step = 128 #256

epochs = 1000
batch_size = 32
latent_dim = 100

SR_output = 16_000 # 44100
d_loss, d_acc, gen_loss = [], [], []

datasetAudio = tf.keras.utils.audio_dataset_from_directory(
    '/content/drive/MyDrive/audios/audio/',
    labels = None,
    label_mode = None,
    #sampling_rate = 44100,
    batch_size = batch_size,
    shuffle = True,
    output_sequence_length = SR_output
)

print("datasetAudio")
print(datasetAudio)


def squeeze(audio):
  audio = tf.squeeze(audio, axis =- 1)
  return audio

datasetAudio = datasetAudio.map(squeeze, tf.data.AUTOTUNE)

def get_spectrogram(waveform):
  spectrogram = tf.signal.stft(waveform, frame_length = frame_length,
frame_step = frame_step)
  spectrogram = tf.abs(spectrogram)
  spectrogram = spectrogram[..., tf.newaxis]
  return spectrogram

def make_spec_ds(ds):
  return ds.map(
      map_func = lambda audio: (get_spectrogram(audio)),
      num_parallel_calls = tf.data.AUTOTUNE)

datasetSpectrums = make_spec_ds(datasetAudio)
print("spectrums")
print(datasetSpectrums)

for example_spectrograms in datasetSpectrums.take(1):
```

```
  break

input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)


generator = tf.keras.Sequential([
    layers.Dense(256, input_shape = (latent_dim,), activation =
'relu'),
    layers.BatchNormalization(),
    layers.Dense(512, activation = 'relu'),
    layers.BatchNormalization(),
    #layers.Dense(784, activation = 'tanh'),
    layers.Dense(118 * 513, activation = 'tanh'),
    layers.Reshape((118, 513, 1))
])
generator.summary()

discriminator = tf.keras.Sequential([
    layers.Flatten(input_shape = (118, 513, 1)),#337*513
    layers.Dense(512, activation = 'relu'),
    layers.Dense(256, activation = 'relu'),
    layers.Dense(1, activation = 'sigmoid'),
    #layers.Dense(1, activation = 'sigmoid'),
])
discriminator.summary()


discriminator.compile(optimizer =
tf.keras.optimizers.Adam(learning_rate = 0.0002),
                      loss = tf.keras.losses.BinaryCrossentropy(),
                      metrics = ['accuracy'])

discriminator.trainable = False
gan_input = tf.keras.Input(shape = (latent_dim,))
gan_output = discriminator(generator(gan_input))
gan = tf.keras.Model(gan_input, gan_output)
gan.compile(optimizer = tf.keras.optimizers.Adam(learning_rate =
0.0002),
            loss = tf.keras.losses.BinaryCrossentropy())

for epoch in range(epochs):
  print(f"Epoch {epoch + 1} / {epochs}")
  for real_images in datasetSpectrums:
    try:
      # Train discriminator
```

```
      random_latent_vectors = tf.random.normal(shape = (batch_size,
latent_dim))
      generated_images = generator.predict(random_latent_vectors)
      combined_images = tf.concat([real_images, generated_images],
axis = 0)
      labels = tf.concat([tf.ones((batch_size, 1)),
tf.zeros((batch_size, 1))], axis = 0)
      discriminator_loss =
discriminator.train_on_batch(combined_images, labels)
      d_loss.append(discriminator_loss[0])
      random_latent_vectors = tf.random.normal(shape=(batch_size,
latent_dim))
      misleading_labels = tf.ones((batch_size, 1))
      generator_loss = gan.train_on_batch(random_latent_vectors,
misleading_labels)
      gen_loss.append(generator_loss)

    except ValueError:
      print('***********\n Error \n***********')

    # Print progress
  print(f"Discriminator Loss: {discriminator_loss[0]} | Discriminator
Accuracy: {discriminator_loss[1]}")
  print(f"Generator Loss: {generator_loss}")

  plt.plot(d_loss, color='blue')
  plt.plot(gen_loss, color='red')
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['d_loss', 'g_loss'], loc='upper left')

    # Generate and save samples
  if epoch % 101 == 0:
    print("Audio_Generado!")
    random_latent_vectors = tf.random.normal(shape = (1, latent_dim))
    generated_spectrum = generator.predict(random_latent_vectors)
    generated_spectrum = tf.squeeze(generated_spectrum, axis = [0,3])
    generated_spectrum = tf.make_tensor_proto(generated_spectrum)
    generated_spectrum = tf.make_ndarray(generated_spectrum)
    print(generated_spectrum.shape)
    audio_signal = librosa.griffinlim(generated_spectrum)
    sf.write(f"/content/drive/MyDrive/data/{epoch}SC.wav",
audio_signal, SR_output, 'PCM_24')


plt.show()
```

7.2 IMAGENES

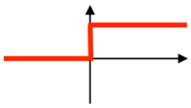
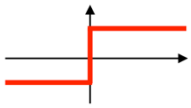
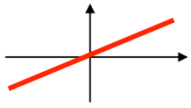
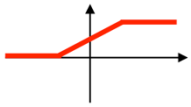
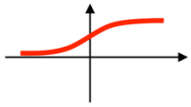
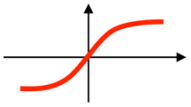
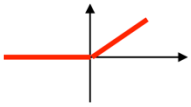
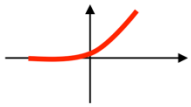
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Figura 22. Funciones de activación para redes neuronales artificiales. Recuperado de https://sebastianraschka.com/faq/docs/activation-functions.html [consultado el 1 marzo de 2024]

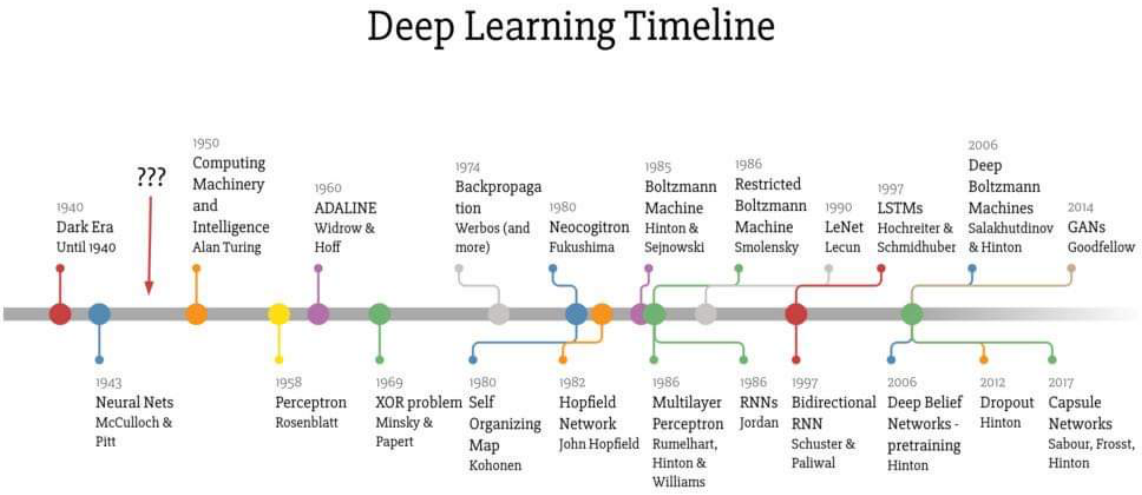


Figura 23. Linea de tiempo de modelos de Aprendizaje Profundo. Recuperado de: https://ar5iv.labs.arxiv.org/html/1908.02130 [consultado el 6 de marzo de 2024]

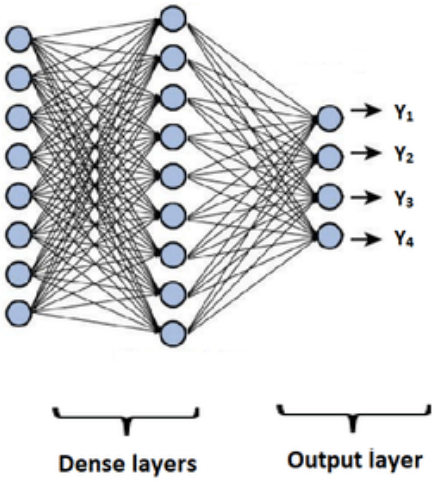
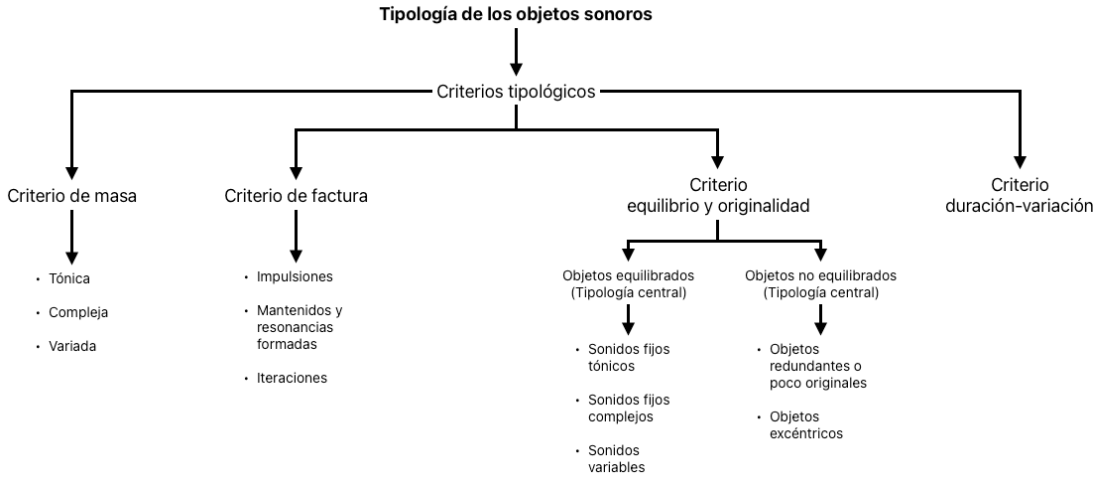
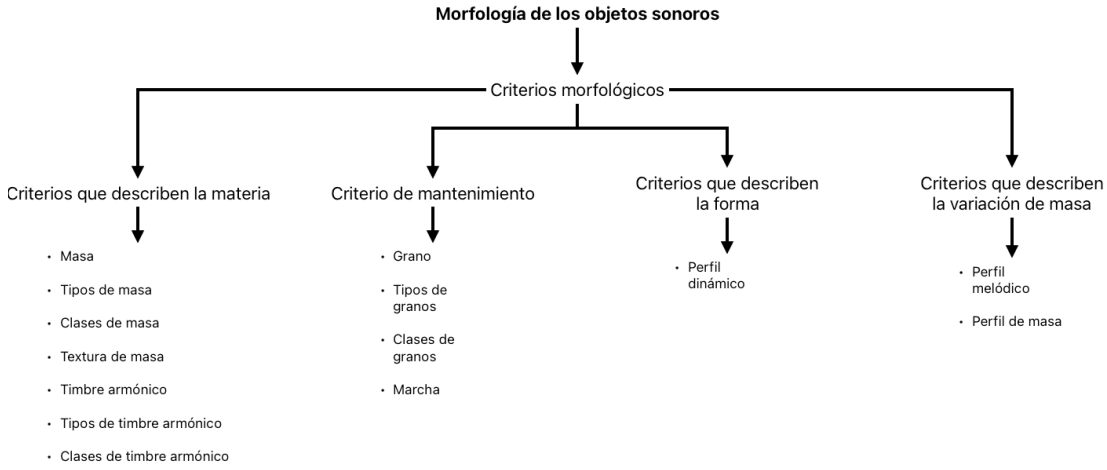


Figura 24. Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases - Scientific Figure on ResearchGate. Recuperado de https://www.researchgate.net/figure/Representation-of-the-architecture-of-a-convolutional-neural-network-CNN_fig2_339278442 [consultado el 2 de febrero de 2024]



Mapa 2. Tipología de los objetos sonoros



Mapa 1. Morfología de los objetos sonoros